Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра Информатики

УТВЕРЖДАЮ
Заведующий кафедрой

_____ _____
подпись инициалы, фамилия
«____» _____ 2017 г.

БАКАЛАВРСКАЯ РАБОТА

27.03.03 — Системный анализ и управление

Алгоритм контроля распространения событий

Руководитель _____ _____ Даничев А.А.
подпись, дата должность, ученная степень

Выпускник _____ Никониров Г.В.
подпись, дата

Красноярск 2017

РЕФЕРАТ

Выпускная квалификационная работа по теме «Алгоритм контроля распространения событий» содержит 40 страниц текстового документа, 6 используемых источников, 12 иллюстраций, 2 таблицы.

ГРАФ, СЮЖЕТ, ГЕНЕРАЦИЯ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, АЛГОРИТМ, КОНТЕКСТ.

Целью данной работы является разработка алгоритма контроля распространения событий для настольной игры.

Актуальность данной работы проявляется в востребованности генерации качественного и проработанного сюжета в современных играх.

Объектом исследования является сюжет в настольной игре.

Предметом исследования является генерация сюжета в настольной игре.

Задачами исследования являются:

1. Исследование предметной области;
2. Разработка алгоритма генерации сюжета;
3. Апробация алгоритма на настольной игре;
4. Анализ полученных результатов.

Апробация алгоритма была произведена при помощи создания программного обеспечения для клуба настольных игр, для моделирования поведения монстров в лесу.

В результате проведения данной работы была изучена литература по данной теме и разработан, а так же апробирован алгоритм контроля распространения событий.

В итоге алгоритм показал отличную работу, а также были сделаны выводы по эффективности и применимости алгоритма в других играх.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В 21 веке распространение персональных компьютеров, а также сложных вычислительных устройств (смартфоны, планшеты и т.д.) достигло своего апогея. Почти у каждого человека есть доступ к ЭВМ. На данный момент компьютеры задействованы во всех сферах человеческой деятельности.

Из-за развития технологий сфера развлечений произвела переход на персональные компьютеры. На данный момент существует огромное множество компьютерных игр. Компьютерные игры имеют множество жанров, но тем не менее почти во всех играх присутствует сюжет [1] или его подобие.

Однако, несмотря на это в некоторых играх сюжет не важен, или не нужен, например, в следующих играх сюжета нет, но это не мешает быть им популярными:

1. World of Tanks;
2. Conter-Strike;
3. Minecraft;
4. Dota 2;
5. EuroTruck Simulator.

Также есть игры, где сюжет играет важную роль, и без него игра не представляет особого интереса для большинства игроков, следующие игры тому пример:

1. World of Warcraft (все версии);
2. Bioshock Infinity;
3. Borderlands (1, 2, Pre-Siquel);
4. Fallout;
5. Mafia II;
6. Mass Effect.

Конечно в них можно играть, не обращая внимания на сюжет, но тогда теряется чувство погружения в игровой мир, что уменьшает удовольствие от игры.

К сожалению для игроков во всех играх, где есть сюжет, интерес к игре (удовольствие от игры) пропадает после пары прохождений. Эту проблему разработчики игр пытаются решить по-разному: кто-то выпускает продолжения серии (Mass Effect), кто-то новые патчи (World of Warcraft). Однако этот процесс достаточно медленный и может занимать пару лет, за которые игра уже может надоесть игрокам.

Наибольшие успехи в развитии сюжета присутствуют в жанре MMORPG (massively multiplayer online role-playing game). Там патчи (обновления сюжета) могут выходить раз в полгода, а иногда и чаще. Но при этом повлиять на развитие сюжета игроки не могут.

Поэтому игрокам интересна игра, где был бы постоянно генерирующийся сюжет, на который может повлиять сам игрок. Чтобы этого добиться необходимо, чтобы в игре происходили некоторые события, которые могут быть вызваны игроком или игрой (миром игры, например монстром из игры), и при этом могут изменить мир игры и повлиять на последующие события.

1 Теоретическая часть

1.1 Актуальность работы

Алгоритм контроля распространения событий представляет собой некоторый алгоритм создания, настройки и работы с системой. Создание такого алгоритма несомненно является работой в области системного анализа [2], ведь системный анализ это вид научных исследований, направленных на изучение системы, выявление взаимосвязей в системе.

Ценность такого алгоритма с точки зрения системного анализа состоит в том, что он позволит создавать взаимосвязанные события в системе, которые при этом не выбиваются за рамки здравого смысла (события бут происходить в рамках разумного и не будут нарушать ограничения системы).

Однако алгоритм представляет ценность еще и с точки зрения компьютерного игропрома (игровая промышленность). Основная ценность состоит в том, что он позволит создавать взаимосвязанный и логичный сюжет в системе (игре) автоматически, при этом реагируя на действия игроков.

1.2 Цели и задачи алгоритма

Для алгоритма можно выделить две основные цели:

1. Генерация сюжета игры на основе происходящих в игре событий;
2. Взаимодействие событий, происходящих в игре на саму игру.

На основе этих целей можно выделить следующие задачи для алгоритма:

1. Алгоритм должен обрабатывать влияние игроков на систему, при этом каждый игрок может оказывать уникальное влияние на систему;
2. События в игре должны быть взаимосвязанными (событие не может появится просто так, ему должно предшествовать действие или другое событие);
3. События должны оказывать влияние на игру (извержение вулкана возле деревни должно ее уничтожать);
4. Алгоритм должен по событиям генерировать сюжет или историю игрового мира.

Однако, для бакалаврской работы полностью реализовать все цели и задачи не представляется возможным из-за большой сложности алгоритма, и наличия неизвестных заранее «подводных камней».

1.3 Аналоги алгоритма

На данный момент найти аналоги алгоритма не удалось. Однако, было выделено несколько компьютерных игр, где используется модель генерации событий, и был произведен анализ плюсов и минусов моделей в этих играх.

1.3.1 Heroes of might and magic

Это известная многим игрокам пошаговая компьютерная игра [3]. Алгоритмически интересна генерацией недели событий (каждые 7 игровых дней основное игровое событие меняется). Примером таких событий может являться неделя гоблинов, это событие значительно увеличивает спаун (появление контролируемых системой монстров) гоблинов.

Плюсами этой системы являются:

1. Некоторое разнообразие геймплея (игрового процесса).

Минусами этой системы является:

1. Отсутствие возможности напрямую повлиять на генерацию события;
2. Ограниченность набора событий;
3. Несвязность появления событий (с более поздними версиями игры частична была уменьшена).

1.3.2 Slaves to Armok II: Dwarf Fortress

Более известная как просто Dwarf Fortress [4], компьютерная игра в жанре «рогалик» (основные характеристики жанра: генерируемые случайно уровни, пошаговый геймплей, конец игры при смерти персонажа). Представляет алгоритмический интерес предварительной генерацией мира. Игрок предварительно настраивает опции генерации мира, и по ним создается мир, при этом во время моделирования процесса создания мира (генерации)

происходят различные события влияющие на результат генерации мира. Также представляет интерес моделирование системой событий после генерации мира (во время игрового процесса).

Плюсами системы является:

1. Разнообразие игровых миров (почти невозможно создать 2 одинаковых мира);
2. Моделирование системой огромного множества аспектов (например реальное воздействие повреждений на существе).

Минусами системы является:

1. Рандомность (проявляются случайно) большинства событий;
2. Ограниченность событий: в созданном мире набор событий ограничен (хотя количественно может быть огромен).

1.4 Предметная область работы

Для работы алгоритма необходимы некоторые данные, в данном случае данные были взяты из моделирования абстрактного процесса настольной ролевой игры (для конкретной игры процесс будет отличатся только данными): прохождение игроком некоторой зоны населенной монстрами. На двухмерной карте находятся гнезда монстров, игроки. Также на зону действует некоторая погода, которая влияет на вероятность атаки монстра. Причем у разных монстров эта вероятность разная. Также погода меняется тоже с некоторой вероятностью и зависит от предыдущей погоды.

1.4.1 Погода

Для всей зоны одновременно действует общая погода, и через некоторое время (количество ходов) она должна меняться. Для смены погоды есть определенный интервал: минимальное число ходов, через которое она может смениться, и максимальное число ходов, через которое она точно сменится. Однако, при этом возможна смена погоды на ту же самую.

На выбор погоды при смене влияет вероятность смены базовая и вероятность смены послепогодная (обе задаются пользователем). Базовая и

послепогодная вероятность нормированы: базовая дает в сумме 1, послепогодная 0 для каждой погоды.

Чтобы получить итоговую погоду:

1. Обе вероятности суммируются;
2. Из вероятностей собирается отрезок длиной 1;
3. По равномерному закону генерируется число от 0 до 1;
4. Интервал на который приходится полученное число и есть искомая погода.

Таблица 1.1 — Пример базовых вероятностей погоды

Погода	Базовая вероятность
Ясно	0.5
Туман	0.2
Дождь	0.2
Шторм	0.1

В таблице 1.1 представлены 4 вида погоды и их базовые вероятности, сумма которых дает 1. В таблице 1.2 представлены их послепогодные вероятности, где строка это текущая погода, а столбец — погода, на которую влияет текущая. Послепогодные вероятности нормированы построчно.

Таблица 1.2 — Пример послепогодных вероятностей

Погода текущая\ погода	Ясно	Туман	Дождь	Шторм
Ясно	0	0	0	0
Туман	-0.2	0	0.2	0
Дождь	0	0.2	-0.3	0.1
Шторм	0	0.2	0.2	-0.4

1.4.2 Монстры

По всей зоне распределены гнезда монстров. Их координаты и характеристики задаются пользователем. Гнездо после каждой атаки снижает

свою активность на определенное число ходов (проявляется в снижении вероятности атаки). Также на разных расстояниях от гнезда различная вероятность быть атакованным. Эта вероятность выражена в виде радиуса вокруг гнезда и вероятности атаки монстра в этом радиусе. При этом вероятность быть атакованным монстром меняется в зависимости от погоды: для каждого гнезда по-своему (например нежить предпочитает атаковать во время тумана, а волки при ясной погоде). Также гнездо монстров может быть разрушено игроком, и тогда активность монстров этого гнезда в течении определенного числа ходов спадает до 0 (гнездо полностью разрушается). В итоге вероятность быть атакованным монстром вычисляется по формуле (1.1)

$$p_a = p_r * k_w \quad , \tag{1.1}$$

где p_r — вероятность атаки монстром для радиуса в котором находится игрок;

k_w — коэффициент влияния текущей погоды на монстра.

В случае если монстр атаковал (вероятность атаки снижена), то формула (1.1) принимает следующий вид

$$p_a = p_r * k_w - p_{aa} \quad , \tag{1.2}$$

где p_r — то же, что и в формуле (1.1);

k_w — то же, что и в формуле (1.1);

p_{aa} — снижение вероятности атаки.

Если гнездо разрушено, то вероятность вычисляется по формуле (1.3)

$$p_{da} = \frac{p_a * c_{dp}}{c_{tdp}} \quad , \qquad (1.3)$$

где p_a — вероятность атаки, посчитанная по формуле (1.1 или 1.2);

c_{dp} — количество ходов оставшихся до полного уничтожения гнезда;

c_{tdp} — количество ходов, необходимых для полного уничтожения гнезда.

1.5 Алгоритм контроля распространения событий

Алгоритм контроля распространения событий строится на системе, являющейся некоторой исследуемой моделью, которая состоит из данных и их обработчика, а также некоторых простых алгоритмов обработки данных.

1.5.1 Данные

Данные в системе представлены в виде ориентированного графа.

Узлами на этом графе будут некоторые элементы, которые могут содержать данные по предметной области, если элемент содержит данные только по одной предметной области, то он называется контекстным и входит в контекстное дерево.

Контекст — это некоторые данные из какой-либо предметной области.

Контекстное дерево — граф (подграф) со структурой дерева, на котором собраны все узлы по некоторой предметной области.

В контекстных деревьях отношение родитель-потомок будет означать, что одно порождено другим, например, на политическо-территориальном контекстном дереве родитель — Россия, потомок - Красноярский край.

В системе будет присутствовать специальное контекстное дерево для хранения событий.

Событие — некоторое действие, происходящие в модели (например война, извержение вулкана и т.д.).

Дугами будут однонаправленные смысловые связи в контексте или между произвольными узлами. Дуги имеют некоторый тип, а также могут иметь некоторую передаточную функцию.

Основные типы дуг:

1. Родитель (parrent) — показывает, что начальный узел является родителем конечного узла;
2. Потомок (child) — показывает, что начальный узел является потомком конечного узла;
3. Событие (event) — показывает, что начальный узел является событием для конечного узла;
4. Контекст (scope) — показывает, что начальный узел связан каким то контекстом с конечным узлом.

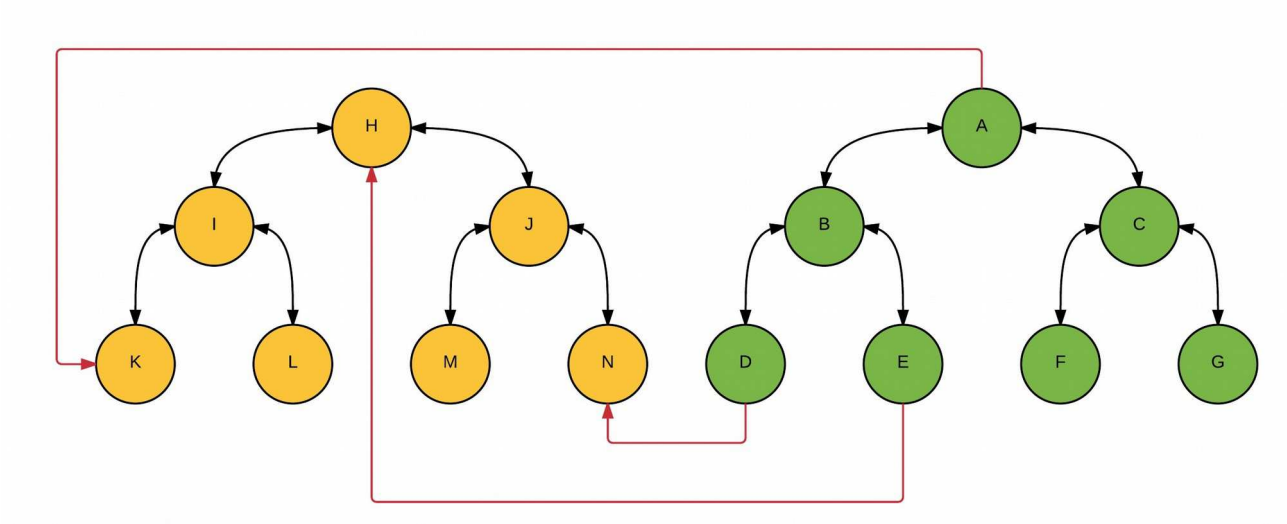

Рисунок 1.1 — Пример графа данных

На рисунке 1.1 изображен граф с данными разными цветами помечены узлы разных контекстных деревьев, черными стрелками связи родитель-потомок, красными связи типа событие.

1.5.2 Обработчик

Обработчик в системе представлен в виде очереди процессов.

Процесс — это некоторое действие, порождаемое событием (извержение вулкана порождает выбросы лавы).

Процесс — это некоторый алгоритм обхода графа с возможным изменением данных и/или изменением структуры графа (добавлением удалением дуг узлов).

Контекстное время — время, используемое в системе (например, секунды, ходы и т.д.).

Процесс является атомарным действием, которое производится не более чем за минимальную единицу контекстного времени. При моделирование он считается мгновенным действием.

У процесса имеются следующие характеристики:

1. Время его исполнения (начала процесса) — время от момента начала моделирования (начала времени в системе) до запуска процесса;
2. Приоритет — значимость процесса, влияет на порядок выполнения процессов в случае одинакового времени исполнения, чем выше, тем раньше будет исполнен процесс;
3. Алгоритм обработки данных — некоторый алгоритм изменения данных уникальный для каждого типа процесса, может использовать более простые алгоритмы для обработки данных.

Очередь процессов представляет собой упорядоченный набор процессов по следующим правилам:

1. Чем раньше время исполнения, тем ближе к началу очереди находится процесс;
2. В случае если у нескольких процессов одинаковое время исполнения, то ближе к началу очереди окажется процесс с более высоким приоритетом;
3. В случае если у нескольких процессов одинаковое время исполнения и приоритет, то ближе к началу очереди будет процесс добавленный в очередь раньше.

1.5.3 Предыдущая структура

До использование структуры данных с несколькими контекстными деревьями использовалась структура с одним контекстным деревом и с большим количеством дуг.

Это контекстное дерево представляло собой политическо-территориальное дерево, узлы которого представляли собой некоторое разбитие по политическим регионам, городам, странам. В этой структуре дуги служили для хранения некоторых данных и связей, а также события были представлены дугами.

Однако, это было крайне не удобно и противоречило логике, ведь данные узлов хранились в дугах. Также работа с данными была крайне неудобной из-за того, что конкретные внутренние данные (например, протяженность дороги) хранились в дуге а не узле.

Чтобы решить эти недостатки, было принято решение использовать несколько контекстных деревьев, включая дерево событий. Это позволило перенести все данные в узлы, оставив в дугах только специфичные данные, вроде коэффициентов функций связывающих узлы.

1.5.4 Достоинства и недостатки алгоритма

Алгоритм, как и любой метод, имеет свои достоинства и недостатки.

Достоинства алгоритма:

1. Универсальность представления данных — позволяет представить почти любые данные в алгоритме;
2. Простой перенос математических моделей в систему — поскольку данные в системе представлены в виде графа, перенос математической модели представляет собой описание структуры графа в алгоритме;
3. Возможность точной настройки структуры данных — алгоритм предусматривает работу с подграфами, разряженными графами, а также графами с различной степенью связности (включая несвязанные графы).

Недостатки алгоритма:

1. Сложность структуры — из-за гибкости алгоритма структура всего графа данных может быть запутана и сложна (могут появляться циклы, зоны разной степени связности);
2. Предварительная настройка — для работы алгоритма необходимо описать структуру данных, а также настроить (или создать) шаблоны

необходимых процессов (событий), это может оказаться весьма долгим и трудоемким процессом;

3. Внесение данных — для работы алгоритма необходимо внесение всех данных, используемых в модели, это может оказаться очень трудоемким процессом в случае использования комплексного моделирования (по нескольким предметным областям).

В итоге можно выделить самое главное достоинство: это возможность охватить произвольное число сфер моделирования. Это достигается гибкостью структуры данных, а также возможностью точной настройки процессов. Однако, плата за эту возможность проявляется в виде достаточно трудной настройки системы. Чем шире область моделирования (или при большом числе областей моделирования), тем больше требуется вносить данных. Чем больше возможных событий может происходить в системе, тем большее число процессов и взаимосвязей между ними необходимо настраивать в системе.

Однако, настройка всех процессов требуется только при их первом создании. После настройки процессы можно переносить на другие модели с незначительными правками или вовсе без них.

1.6 Простые алгоритмы обработки данных

Это некоторые базовые алгоритмы, которые могут использоваться всеми процессами.

1.6.1 Нахождение корня в контекстном дереве

Этот алгоритм находит корень в контекстном дереве по любому из узлов этого дерева.

Действия алгоритма:

1. Среди всех дуг узла найти дугу с типом потомок, если такой нет - то корень найден, алгоритм завершает работу;
2. Перейти в конечный узел этой дуги;
3. Повторить пункты 1-2.

1.6.2 Перерасчет уровней узлов в контекстном дереве

Этот алгоритм определяет уровень (расстояние до корня) контекстного дерева для каждого из его узлов.

Действия алгоритма:

1. Найти корень дерева (используя алгоритм нахождения корня);
2. Присвоить корню уровень 0;
3. Среди всех дуг узла найти дуги с типом родитель;
4. Перейти в конечный узел дуги;
5. Присвоить узлу уровень его родителя+1;
6. Повторить пункты 4-5 для каждой дуги;
7. Повторить пункты 3-6 для каждого узла;
8. Алгоритм закончил работу.

1.6.3 Поиск наименьшего общего предка

Этот алгоритм для двух узлов контекстного дерева находит наименьшего общего предка (первый узел общий по пути к корню).

Действия алгоритма:

1. Для каждого из двух узлов построить путь к корню;
2. Для двух путей выделить общую часть;
3. Найти на общей части самый удаленный от корня узел, он и будет наименьшим общим предком.

1.7 Использование алгоритма в сторонних приложениях

Помимо возможности создания надстройки приложения над программой (реализацией алгоритма в виде программного обеспечения) существует еще несколько способов использования приложения с другим программным обеспечением (рисунок 1.2).

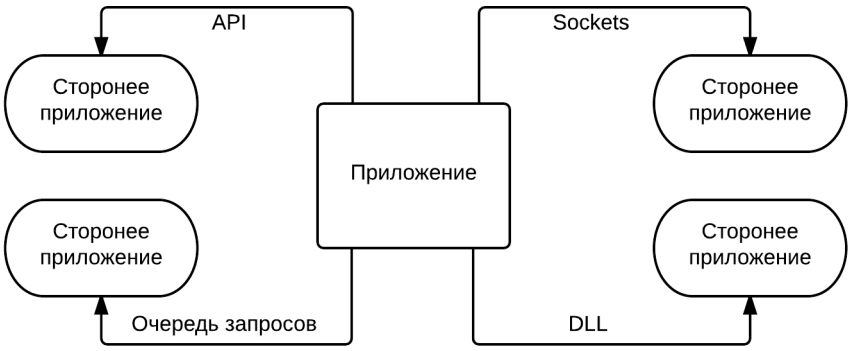

Рисунок 1.2 — Соединение приложения с другими

Разберем каждый из способов подробнее.

1.7.1 Соединение посредством API

API общий термин для соединения стороннего приложения с другим посредством использования вызовов функций и т. п. исходного приложения. Если быть более точным, то в итоге получается единое приложение, которое использует реализованный функционал нашего приложения, а также возможно изменяет его. В итоге стороннее приложение представляет собой модифицированную версию нашего приложения.

У этого соединения есть достоинства:

1. Если пользователю требуется изменение работы некоторых функций, то он может достаточно просто (не требуется создание лишнего функционала для изменения поведения функций) модифицировать их как ему нужно;
2. Пользователь может добавлять нужный функционал по своему усмотрению.

Однако, у соединения также есть и свои минусы:

1. Для использования такого соединения необходимо знать принципы работы исходного приложения, а также его архитектуру; на получение необходимых знаний может потребоваться продолжительное время;
2. Изменение алгоритмов работы исходного приложения (модификация) может повлечь появление ошибок в работе программы.

1.7.2 Соединение посредством DLL

DLL общий термин для использования одного приложения другим как библиотекой. Немного похож на соединение посредством API, однако, отличается тем что, не позволяет модифицировать исходное приложение и использует его как есть.

У этого метода есть достоинства:

1. Работа библиотеки (исходного приложения) стабильна, т. е. не вызывает ошибок в программе;
2. В библиотеке, как правило, реализовано множество методов, это освобождает пользователя от их самостоятельного создания.

Однако, этот метод имеет и недостатки:

1. Отсутствие возможности изменения работы алгоритмов из библиотеки, и чтобы их модифицировать приходится использовать методы обходящие работу библиотеки (или нарушающие ее), что может привести к некорректной работе приложения;
2. Стабильность и качество работы библиотеки напрямую зависит от ее разработчика.

1.7.3 Соединение посредством очереди запросов

Очередь запросов это некоторый набор запросов, отправляемых в исходное приложение, с возможным получением некоторого ответа. В целом этот метод отдаленно напоминает DLL, но отличается от него тем, что исходному приложению отправляются запросы, при этом не производится вызов функций. Если быть более точным, то вызовы функций все-таки производятся, но их вызовом управляет исключительно исходное приложение, и вмешаться в их вызов невозможно. Таким образом, можно сравнить этот метод с диалогом: мы разговариваем с собеседником (посылаем ему запросы), на что он нам может ответить.

Это соединение имеет ряд достоинств:

1. Для использования исходного приложения не нужно знать его архитектуры, достаточно знаний запросов, которые ему можно отправлять;

2. Ошибки, появившиеся в результате работы исходного приложения не приводят к аварийному завершению работы стороннего приложения.

Также у соединения есть свои минусы:

1. Изменить работы алгоритмов исходного приложения невозможно;
2. Качество и стабильность работы исходного приложения зависит только от его разработчика.

1.7.4 Соединение посредством Sockets

Технология соединения TCP/IP Sockets [5] является весьма распространенной, и используется в большинстве сетевых приложений (использующих для своей работы технологию internet). Эта технология является сильно модифицированным соединением посредством очередей. Основное отличие заключаются в том, что одно из приложений играет роль сервера, при этом в отличии от предыдущих типов соединений к серверу могут подключатся несколько клиентов (сторонних приложений).

Достоинства соединения:

1. Возможность работать сразу с нескольким сторонними приложениями;
2. Исходное приложение может быть размешено где угодно, для работы требуется только наличие соединения.

Однако есть так же и минусы:

1. Настройка исходного приложения как сервера очень трудна;
2. Без наличия сетевого соединения между приложениями это соединение не работает;
3. Скорость работы зависит от скорости соединения.

2 Практическая часть

Для реализации системы был выбран язык программирования python [6]. Этот язык является наиболее подходящим, т.к. он кроссплатформенный и запускается через виртуальную машину, что облегчает интеграцию системы, написанной на этом языке с другими системами и приложениями. В качестве метода для соединения со сторонними приложениями был выбран метод API.

2.1 Архитектура приложения

Архитектура приложения состоит из архитектуры модулей и архитектуры классов.

Сначала рассмотрим архитектуру модулей.

2.1.1 Архитектура модулей

Поскольку язык питон поддерживает модульную архитектуру приложений, было решено использовать все ее достоинства, а именно, группировку различных частей приложения по модулям, чтобы обеспечить тематическую группировку программного кода, а также сохранить целостность всей системы.

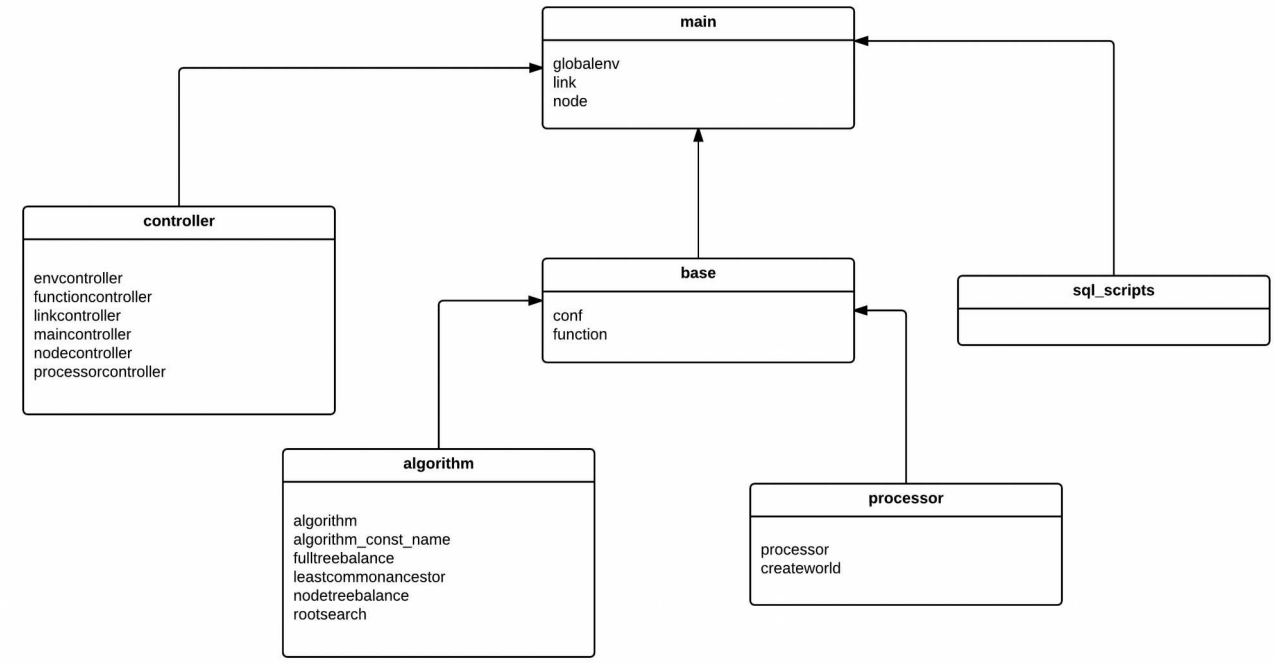

Рисунок 2.1 — Схема модулей

На рисунке 2.1 изображена общая схема модулей, на ней блок обозначает составной модуль (модуль из нескольких подмодулей), текст внутри блока это список простых подмодулей этого модуля, стрелка обозначает, что модуль является подмодулем другого модуля. Рассмотрим назначения каждого составного модуля в отдельности.

2.1.1.1 Модуль main

Основной модуль приложения. Запуск приложения производится запуском этого модуля.

Содержит простые модули:

1. globalenv — модуль для работы с глобальными данными переменными и т. д.;
2. link — модуль с классом связей (дуг), а также дополнительными данными для его работы;
3. node — модуль с классом узлов, а также дополнительными данными для его работы.

2.1.1.2 Модуль controller

Модуль приложения содержащий набор контроллеров для управления приложением.

Содержит простые модули:

1. envcontroller — модуль с классом контроля окружения. Управляет глобальными переменными, выдачей уникальных номеров объектам системы;
2. functioncontroller — модуль с классом контроля функций дуг;
3. linkcontroller — модуль с классом для управления дугами (создание, удаление, изменение);
4. maincontroller — модуль с классом для управления самим приложением (запуск, остановка, сохранение загрузка и т. д.);
5. nodecontroller — модуль с классом для управления узлами (создание, удаление, изменение);
6. processorcontroller — модуль с классом для управления процессами. Управляет поведением системы.

2.1.1.3 Модуль sql_scripts

Модуль приложения, содержащий набор скриптов для работы с базами данных. Может быть модифицирован программистом использующим приложение.

2.1.1.4 Модуль base

Модуль приложения, содержащий методы и алгоритмы, необходимые для работы системы. Может быть модифицирован программистом, использующим приложение.

Содержит простые модули:

1. conf — модуль с конфигурацией приложения и системы;
2. function — модуль с функциями дуг.

2.1.1.5 Модуль algorithm

Модуль приложения, содержащий алгоритмы для изменения, получения данных (графа). Может быть модифицирован программистом, использующим приложение.

Содержит простые модули:

1. algorithm — модуль с базовым классом алгоритмов, а также данными для его работы;
2. algorithm_const_name — модуль с именами, используемыми в алгоритмах (имена для данных и т. п.);
3. fulltreebalance — модуль с классом для пересчета уровней контекстного дерева;
4. nodetreebalance — модуль с классом для пересчета уровня узла контекстного дерева;
5. leastcommonancestor — модуль с классом для поиска наименьшего общего предка в контекстных деревьях;
6. rootsearch — модуль с классом для поиска корня в контекстных деревьях.

2.1.1.6 Модуль processor

Модуль приложения, содержащий алгоритмы процессов, происходящих в системе и их обработчики (процессоры). Может быть модифицирован программистом, использующим приложение.

Содержит простые модули:

1. processor — модуль с базовым классом процессоров и данными для его работы;
2. createworld — модуль с классом процессора, запускающего моделирование системы.

2.1.2 Архитектура классов

Архитектура модулей позволяет расположить части приложения более эргономично и удобно для использования, тем не менее, работа приложения полностью зависит от архитектуры программного кода, которая представляется в виде архитектуры классов.

На рисунке 2.2 изображена диаграмма классов, на ней можно выделить 2 группы классов: связанные с MainController, и не связанные с ним. Класс _algorithm не связан с MainController, но при этом объекты этого класса могут использоваться в методах класса _processor.

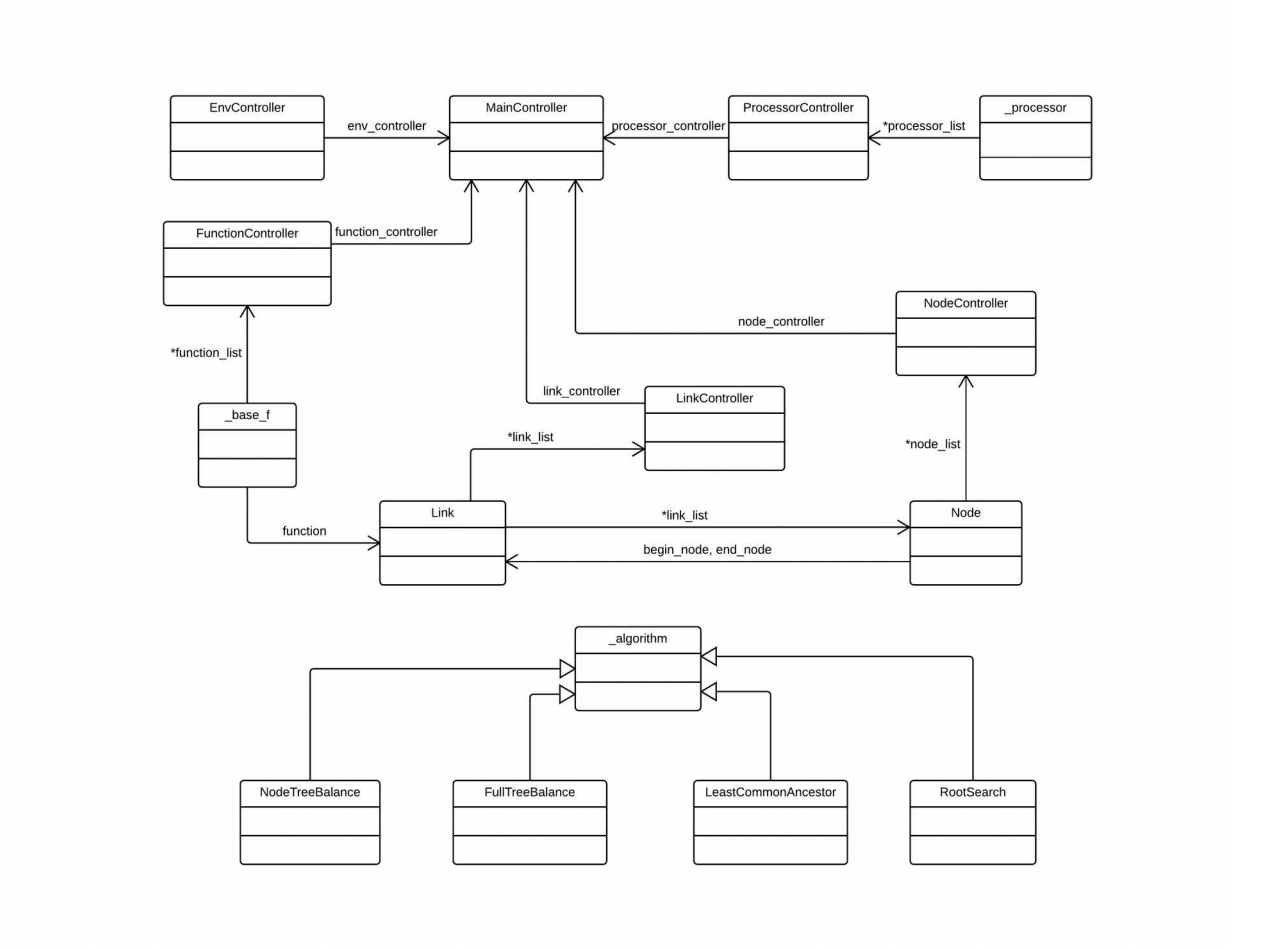

Рисунок 2.2 — UML диаграмма классов

Теперь рассмотрим основные классы более детально (классы наследники классов _algorithm и _processor не будут рассмотрены, т. к. они только переопределяют методы родителя).

2.1.2.1 Класс Link

Класс для представление дуг графа в коде.

Переменные объекта класса:

1. data — хеш таблица дополнительных данных для дуги;
2. function — объект класса _base_f или его наследников;
3. begin_node — объект класса node, являющийся начальным узлом дуги;
4. end_node — объект класса node, являющийся конечным узлом дуги;
5. name — строка, название дуги (задается пользователем);

6. _type — строка, тип дуги (родитель потомок и т. д.);
7. _id — число, уникальный номер дуги в наборе данных.

Методы класса:

1. __init__ — конструктор класса;
2. setConfig — функция для изменения параметров дуги;
3. getConfig — функция для получения параметров дуги;
4. getId — функция для получения _id дуги;
5. getNodes — функция для получения начального и конечного узлов дуги.

2.1.2.2 Класс Node

Класс для представления узлов графа.

Переменные объекта класса:

1. data — хеш таблица дополнительных данных для узла;
2. link_list — список объектов дуг, выходящих из узла;
3. name — строка, название узла (задается пользователем);
4. _type — строка, тип узла (контекст этого узла);
5. _id — число, уникальный номер узла.

Методы класса:

1. __init__ — конструктор класса;
2. setConfig — функция для изменения параметров узла;
3. getConfig — функция для получения параметров узла;
4. getId — функция для получения _id узла;
5. addLink — функция для добавления исходящей дуги;
6. delLink — функция для удаления исходящей дуги.

2.1.2.3 Класс LinkController

Класс для управления дугами в графе.

Переменные объекта класса:

1. link_list — хеш таблица дуг;

2. main_controller — объект класса MainController.

Методы класса:

1. __init__ — конструктор класса;
2. getByType — функция получения списка дуг с указанным типом;
3. getById — функция получения дуги по ее _id;
4. setLinkList — функция создания дуг по их данным (параметрам);
5. getLinkList — функция получения списка параметров дуг;
6. addLink — функция добавления дуги.

2.1.2.4 Класс NodeController

Класс для управления узлами в графе.

Переменные объекта класса:

1. node_list — хеш таблица узлов;
2. main_controller — объект класса MainController.

Методы класса:

1. __init__ — конструктор класса;
2. getById — функция получения узла по его _id;
3. getByType — функция получения списка узлов по их типу;
4. getLinkedNodesByIdLinkType — функция получения узлов связанных с указанным дуга указанного типа;
5. getNodeList — функция получения списка параметра узлов;
6. setNodeList — функция создания узлов по их параметрам;
7. addNode — функция добавления узла.

2.1.2.5 Класс _processor

Базовый абстрактный класс для процессов.

Переменные объекта класса:

1. data — хеш таблица дополнительных данных для процесса;
2. run_time — число, время запуска процесса;
3. _id — число, уникальный номер процесса;

4. _type — строка, тип процесса;
5. weight — число, приоритет процесса.

Методы класса:

1. __init__ — конструктор класса;
2. __lt__ — перегруженный оператор сравнения меньше;
3. setConfig — функция для изменения параметров процесса;
4. getConfig -функция для получения параметров процесса;
5. getId — функция получения _id процесса;
6. getType — функция получения типа процесса;
7. run — абстрактный метод вызова процесса.

2.1.2.6 Класс ProcessorController

Класс для управления процессами.

Переменные объекта класса:

1. processor_list — хеш таблица процессов;
2. current_time — текущее время в модели;
3. main_controller — объект класса MainController.

Методы класса:

1. __init__ — конструктор класса;
2. setConfig — функция изменения параметров контроллера;
3. getConfig — функция получения параметров контроллера;
4. getProcessorList — функция получения списка параметров процессов;
5. setProcessorList — функция создания процессов по их параметрам;
6. addProcessor — функция добавления процесса;
7. createProcessor — функция создания процесса;
8. runOneTik — функция моделирования системы (моделирует минимальную единицу времени в системе).

2.1.2.7 Класс _base_f

Базовый класс функции дуги.

Методы класса:

1. getId — функция возвращает идентификатор функции дуги;
2. __call__ — перегруженный метод вызова объекта как функции.

2.1.2.8 Класс FunctionController

Класс для управления функциями дуг.

Переменные объекта класса:

1. function_list — хеш таблица функций дуг.

Методы класса:

1. __init__ — конструктор класса;
2. getFunction — функция получения функции дуги по ее идентификатору.

2.1.2.9 Класс EnvController

Класс для управления окружением системы (выдача идентификаторов).

Переменные объекта класса:

1. last_link_id — число, последний номер дуги;
2. last_node_id — число, последний номер узла;
3. last_processor_id — число, последний номер процесса.

Методы класса:

1. __init__ — конструктор класса;
2. getNewLinkId — функция, выдающая новый уникальный номер дуги;
3. getNewNodeId — функция, выдающая новый уникальный номер узла;
4. getNewProcessorId — функция, выдающая новый уникальный номер процесса;

5. setConfig — функция изменения параметров контроллера;
6. getConfig — функция получения параметров контролера.

2.1.2.10 Класс _algorithm

Базовый абстрактный класс алгоритмов.

Переменные объекта класса:

1. controller — объект класса MainController.

Методы класса:

1. __init__ — конструктор класса;
2. run — функция, вызываемая при запуске алгоритма

2.1.2.11 Класс MainController

Класс управляющий приложением.

Переменные объекта класса:

1. env_controller — объект класса EnvController;
2. function_controller — объект класса FunctionController;
3. processor_controller — объект класса ProcessorController;
4. node_controller — объект класса NodeController;
5. link_controller — объект класса LinkController.

Методы класса:

1. __init__ — конструктор класса;
2. load — функция загрузки данных из БД;
3. save — функция сохранения данных в БД;
4. createNode — функция создания нового узла;
5. createLink — функция создания новой дуги;
6. createProcessor — функция создания нового процесса.

2.1.3 Причины выбора архитектуры

Начнем с причин выбора конкретной модульной архитектуры (выбора такого разбиения на модули).

Изначально от разбиения на модули требовалось следующее:

1. Наличие отдельного модуля для работы с базой данных;
2. Наличие отдельного модуля управления приложением;
3. Наличие модуля, который может быть модифицирован пользователем под конкретные алгоритмы, процессы и т. д.;
4. Наличие отдельного модуля для класса _algorithm и его наследников;
5. Наличие отдельного модуля для класса _processor и его наследников;
6. Весь код должен находится в глобальном модуле, который можно модифицировать и (или) использовать как библиотеку.

Чтобы удовлетворить всем этим требованием была разработана архитектура, в которой есть модуль algorithm для класса _algorithm, модуль processor для класса processor. Чтобы удовлетворить потребности к модификации приложения был создан модуль base, который предусматривает модификацию пользователем. Для работы с базой данных был создан отдельный модуль sql_scripts. Для управления приложением был создан модуль controller. А для объединения всего кода был создан один большой модуль main. За счет использования языка программирования python модули могут быть использованы как приложения, и как библиотека.

Выбор такой архитектуры классов обусловлен выбором способа представления данных в приложении.

Для работы приложения требуются данные узлов дуг графа с данными, а также данные запланированных процессов (порожденных событиями). Наиболее очевидным решением является то, что для каждой дуги, узла и процесса требуется отдельный объект класса, а сам класс будет общим

представлением дуги, узла или события (для дуги, узла и процесса используются разные классы).

Прежде всего стоит начать с класса для представления узла. Очевидно, что очень важным будет хранение данных узла, которые заданы или интересуют пользователя, а также используются в приложении. Для этого наиболее оптимальным решением будет использования одного поля типа словарь (хеш таблица). У каждых данных будет свой уникальный ключ, например, название этих данных. При этом значения в словаре могут иметь произвольную структуру, т. е. хранить массивы, другие словари и т. д. Для работы ряда алгоритмов будет необходимо хранить список всех дуг, которые выходят из этого узла, для этого лучше использовать структуру данных словарь, где ключом будет уникальный номер дуги в системе, а его значение - объект класса дуги. Также для некоторых алгоритмов необходимо хранить тип узла в виде строки.

Для класса дуг внутренние данные, в отличии от узла, требуются очень малому количеству алгоритмов (могут вообще не использоваться пользователем), тем не менее они могут потребоваться. Для их представления воспользуемся решением, найденным для класса узлов, создадим словарь. Важным для дуги будут являться узлы, которые она соединяет и ее тип. Для представления узлов достаточно в классе сделать две ссылки: одну на объект начального узла, другую на объект конечного узла. Тип дуги можно хранить в виде обычной строки.

Для класса процессов внутренние данные не менее важны, чем для класса узлов. Для их хранения воспользуемся способом, примененным к узлам, в виде словаря. Также для процесса очень важен его тип, ведь именно он определяет как будет происходить процесс. Для его хранения, как и с ссылками, используем обычную строку. Не менее важны для процесса время его начала и приоритет. Для каждого из них имеет смысл сделать отдельное числовое поле. Также

необходимо в самом классе реализовать возможность сравнения процессов по приоритету и времени начала с возможностью сортировки.

После того как определились со способом представления данных, необходимо подумать о способе их загрузки в программу и сохранения из программы. Для этого подойдет какая-нибудь база данных. Сохранение в нее и загрузку можно реализовать посредством передачи данных приложения в нее через SQL скрипты. Конечная настройка этих скриптов зависит от базы данных и производится пользователем.

Однако, для того чтобы работать с данными, в том числе и подготавливать их для базы данных, нужны контроллеры. Для начала рассмотрим контроллеры для самой работы с данными.

Для узлов необходим свой контроллер, который будет содержать узлы, а также предоставлять некоторые методы для работы с ними (как с некоторым набором данных). Самый очевидный метод — это метод получения узла по его уникальному номеру в системе. Также понадобится метод для получения узлов по их типу. Метод получения корня контекстного дерева для уникального номера одного из его элементов облегчит работу части алгоритмов. Также для некоторых алгоритмов понадобится метод для поиска связанных с указанным узлом узлов дугой определенного типа. Ну и для облегчения загрузки и выгрузки данных понадобятся методы, переводящие объект класса узла в базовые типы (представляющие все данные узла).

Для дуг контроллер также должен содержать все объекты дуг и предоставлять методы для работы с ними. Как и в контроллере узлов понадобятся методы получения дуги по ее номеру, а также набора дуг по их типу. Ну и разумеется не обойтись без аналогичных методов перевода данных.

Для процессов контроллер тоже должен содержать все объекты процессов и иметь методы для работы с ними. Правда, в отличии от контроллеров узлов и

дуг, ему не требуется методы для получения процессов, хотя методы для перевода данных все же потребуются. Также потребуется метод создания нового процесса. Ну и самым важным будет метод, выполняющий моделирование минимальной единицы времени в системе.

Помимо контроллеров самих данных также потребуется контроллер окружения. Его основной задачей является управление уникальными номерами данных в системе. В эту задачу также входит выдача новых номеров и подсчет количества элементов в системе.

Для управления всем приложением потребуется еще один контроллер. В задачи этого контроллера входит загрузка выгрузка данных, добавление новых данных.

2.2 Приложение

Для использования алгоритма конечными пользователями было создано приложение с графическим интерфейсом рисунки 2.3-2.9.

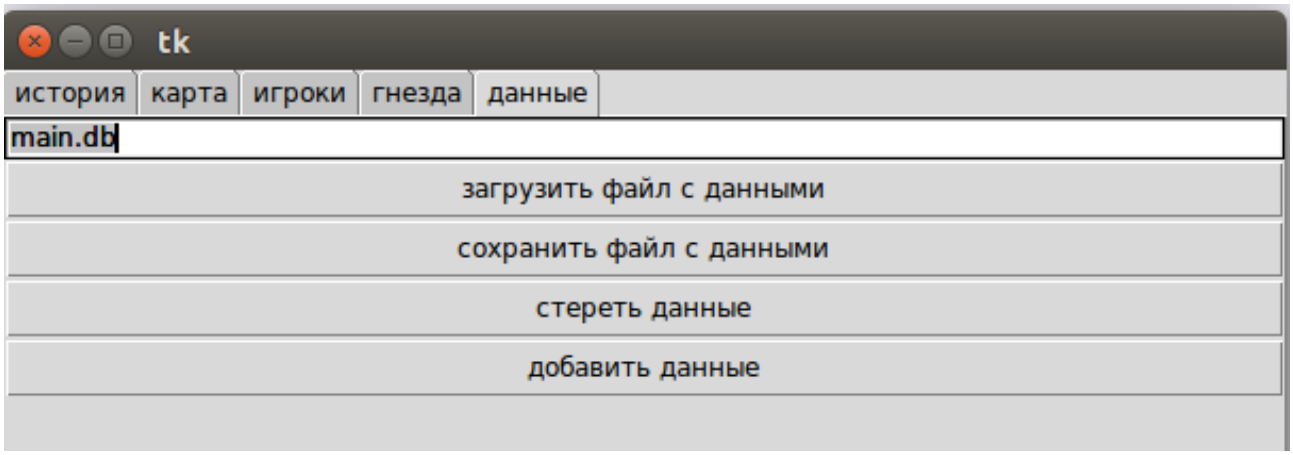

Рисунок 2.3 — Вкладка данные

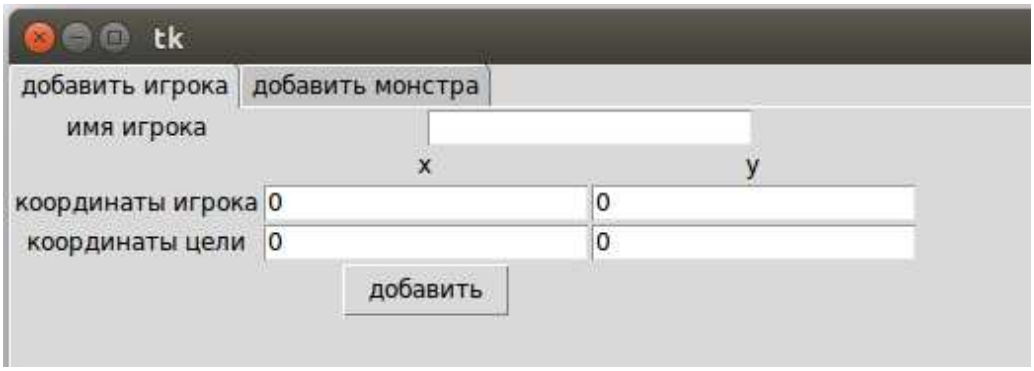

Рисунок 2.4 — Добавление данных вкладка с игроком

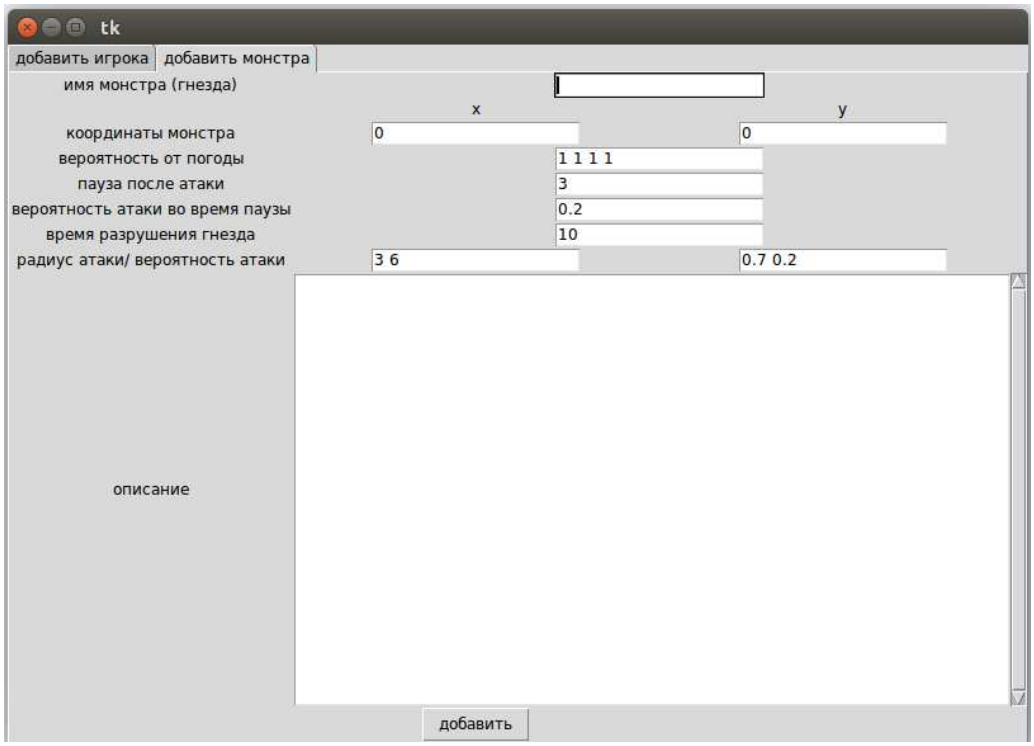

Рисунок 2.5 — Добавление данных вкладка с монстром

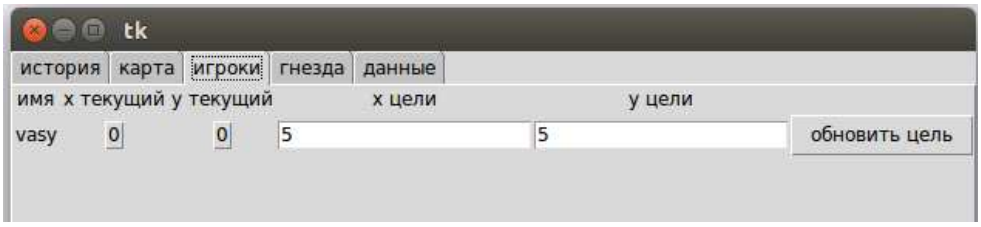

Рисунок 2.6 — Вкладка с текущими игроками

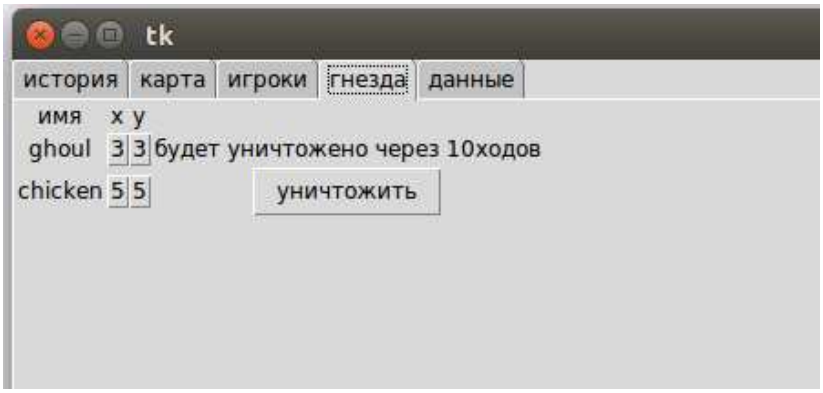

Рисунок 2.7 — Вкладка с текущими гнездами монстров

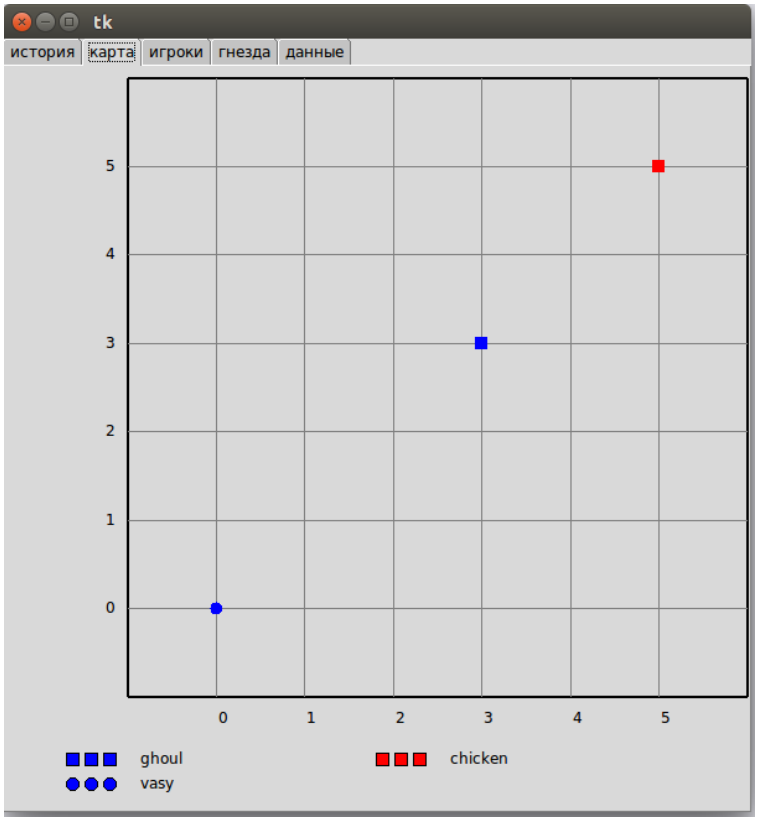

Рисунок 2.8 — Вкладка с картой игровой местности

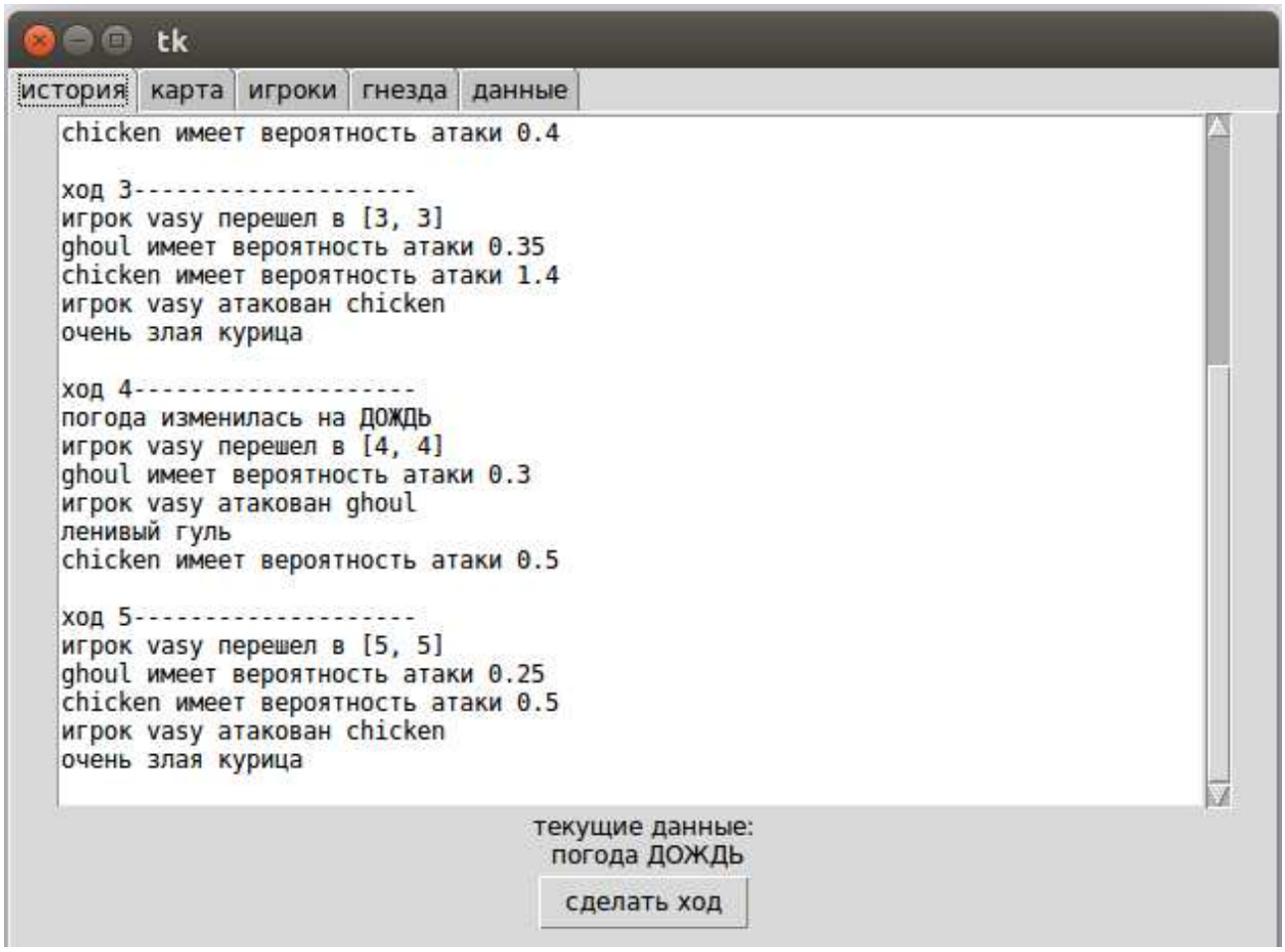

Рисунок 2.9 — Вкладка с игровой историей

Изначально к приложению были следующие требования:

1. Приложение должно иметь возможность сохранять и загружать данные;
2. В приложении должна быть возможность добавлять новые данные;
3. Должна быть возможность уничтожать гнезда монстров;
4. В приложении можно задавать конечные координаты для передвижения игрока;
5. Графическая визуализация ситуации на игровом поле;
6. Генерация истории (отчета действий) за ход.

В результате создания приложения все требования были учтены и реализованы. Пользователям понравился интерфейс приложения, а также его работа.

2.3 Результаты работы алгоритма в приложении

В приложении алгоритм работал с данными, изображенными на рисунке 2.10, на нем зеленым помечены узлы событийного дерева, желтым - дерево игроков, красным - дерево монстров; черные дуги это связи родитель-потомок, красные - связи типа событие.

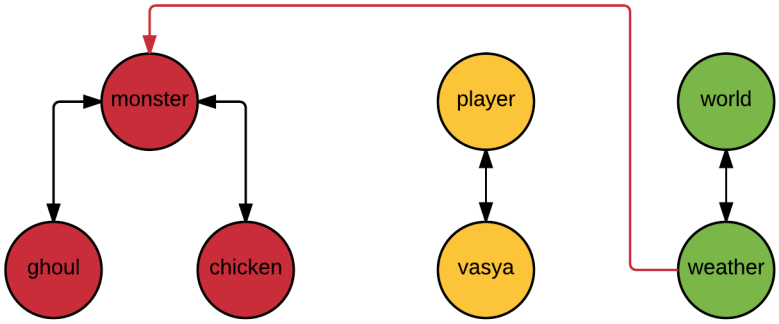

Рисунок 2.10 — Пример графа данных в приложении

Работа алгоритма в приложении была высоко оценена пользователем. Алгоритм показал высокую точность работы, при этом не были нарушены принципы здравого смысла модели. Алгоритм позволил достаточно просто моделировать атаки монстров на игроков, учитывая при этом разнообразные факторы.

Такие высокие результаты работы алгоритма были получены из-за достаточно простой модели и корректно введенных данных. При более сложных моделях результат может быть более низким. Однако конечный результат очень сильно зависит от корректности заданной в программе модели, а также точности используемых при моделировании данных.

ЗАКЛЮЧЕНИЕ

В рамках данной бакалаврской работы был разработан и опробован алгоритм контроля распространения событий.

Также было разработано и протестировано графическое приложение для моделирования атак монстров на игроков в настольных играх на основе алгоритма.

В результате тестирования алгоритм показал свою надежность и точность, а также применимость на настольных играх.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wikipedia [Электронный ресурс] : Сюжет — режим доступа : https://ru.wikipedia.org/wiki/%D0%A1%D1%8E%D0%B6%D0%B5%D1%82

2. Перегудов, Ф. И. Введение в системный анализ : учеб. пособие для вузов / Ф. И. Перегудов, Ф. П. Тарасенко — Москва : Высшая школа, 1989. - 360с

3. Wikipedia [Электронный ресурс] : Heroes of Might and Magic (серия игр) — режим доступа : https://ru.wikipedia.org/wiki/Heroes_of_Might_and_Magic_(%D1%81%D0%B5%D1%80%D0%B8%D1%8F_%D0%B8%D0%B3%D1%80)

4. Bay 12 Games: Dwarf Forrtress [Электронный ресурс] : Dwarf Fortress — режим доступа : http://www.bay12games.com/dwarves/

5. Wikipedia [Электронный ресурс] : Сокет — режим доступа : https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)

6. Python Documentation [Электронный ресурс] : Python 3 Documentation — режим доступа : https://docs.python.org/3/

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра Информатики

УТВЕРЖДАЮ
Заведующий кафедрой
А.С. Кузнецов
подпись инициалы, фамилия
«19» 06 2017 г.

БАКАЛАВРСКАЯ РАБОТА

27.03.03 — Системный анализ и управление

Алгоритм контроля распространения событий

Руководитель 19.06.17 доцент, канд. техн. наук Даничев А.А.
подпись, дата должность, ученная степень

Выпускник 19.06.17 Никониров Г.В.
подпись, дата

Красноярск 2017